

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

UTILITY PATENT APPLICATION FOR:

**METHOD AND APPARATUS FOR COMPILING
SOURCE CODE TO CONFIGURE HARDWARE**

INVENTOR:

GREGORY S. SNIDER
1529 Meadow Lane
Mountain View, California 94040

METHOD AND APPARATUS FOR COMPILING SOURCE CODE TO CONFIGURE HARDWARE

FIELD OF THE INVENTION

The invention relates generally to compiling source code and more particularly to a compiling source code to configure hardware.

BACKGROUND OF THE INVENTION

The advancement of the integrated circuit, printed circuit board and other related technologies is advancing at a very rapid rate. The latest generation of integrated circuits can incorporate over more four times the amount of circuitry that was possible just a few years ago. Furthermore, circuit board and multi-chip module technology have allowed much denser circuit board designs. These and other developments have increased the development of increasingly complex and high-speed computer systems.

The design of such high-speed computer systems has become increasingly difficult and time consuming. In order to maximize performance and to minimize the size and power of such computer systems, designers often implement much of the hardware in a number of integrated circuits. The integrated circuits are often custom or semi-custom designed. Each of these custom integrated circuits may contain several hundred thousand gates, and each gate must be placed and routed in accordance with the overall computer system specification.

To design such a computer system, the designer typically produces an overall system specification using a hardware description language. VHDL and Verilog are the most common conventional hardware description languages. VHDL describes the behavior and structure of electrical systems, but is particularly suited as a language to describe the behavior and structure of digital electronic hardware designs, such as application specific integrated circuits (ASICs) and field programmable gate arrays (FPGAs) as well as conventional digital circuits. Verilog is a textual format for describing electronic circuits and systems. Applied to electronic design, Verilog is intended to be

1 used for system verification through simulation, for timing analysis, for test analysis
2 (testability analysis and fault grading) and for logic synthesis.
3

4 As electronic hardware design becomes increasingly miniaturized, the tools of the
5 designer must allow for more flexibility. The problem with conventional design languages
6 like VHDL and Verilog is efficiently implementing hardware structures for computation. The
7 conventional hardware design languages do not automate the micro-architectural designs
8 needed.
9

10 The described approaches are not able to synthesize many different hardware
11 implementations depending on the global context of the computation without having to
12 change or annotate the original source.
13

14 SUMMARY OF THE INVENTION 15

16 In one embodiment, the invention is a method of implementing operations
17 representing computations in a compiler, the method comprising the steps of parsing a
18 source code, performing a plurality of optimizations on the parsed code, generating a
19 plurality of configuration instruction sets based on the optimized source code, and
20 automatically selecting one of the plurality of generated configuration instruction sets
21 according to a user defined criteria, the selected configuration instruction set being used to
22 configure hardware.
23

24 In another respect, the invention is a method for compiling source code comprising
25 steps of generating an internal representation of the source code, analyzing data flow
26 properties of the internal representation in order to optimize the internal representation,
27 automatically generating a plurality of configuration instruction sets based on the
28 optimized internal representation, generating a plurality of configuration instruction sets
29 based on the optimized source code, and automatically selecting one of the plurality of

1 generated configuration instruction sets according to a user defined criteria, the selected
2 configuration instruction set being used to configure hardware.

3
4 In another respect, the invention is a system for using software to generate a circuit
5 comprising a processor operable to receive source code, a compiler automatically
6 generating a plurality of configuration sets from the received source code and selecting
7 one of the plurality of configuration sets based on user defined criteria, and a configurable
8 hardware device receiving the selected configuration instruction set and being configured
9 based on the received configuration instruction set.

10
11 In comparison to known prior art, certain embodiments of the invention are
12 capable of achieving certain aspects, including some or all of the following: (1) allows the
13 user to design hardware using arithmetic/logical operations without worrying about the
14 implementation (2) hides implementation issues in hardware synthesis from the
15 application ; and (3) creates a layer of abstraction between the hardware and application
16 levels. Those skilled in the art will appreciate these and other advantages and benefits of
17 various embodiments of the invention upon reading the following detailed description of a
18 preferred embodiment with reference to the below-listed drawings.

19 20 BRIEF DESCRIPTION OF THE DRAWINGS

21
22 The invention is described in greater detail hereinafter, by way of example only,
23 through description of a preferred embodiment thereof and with reference to the
24 accompanying drawings in which:

25
26 FIG. 1 is a block diagram illustration of a system of the invention, according to an
27 embodiment of the invention;

28
29 FIG. 2 is a flow-chart illustrating a method performed by a hardware compiler,
30 according to an embodiment of the invention;

1
2 FIG. 3 is a flow-chart illustrating a method of performing mid-level optimizations,
3 according to an embodiment of the invention;
4

5 FIG. 4A is an illustration of an unpipelined multiplier, according to an embodiment of
6 the invention;
7

8 FIG. 4B is an illustration of a micropipelined multiplier, according to an embodiment
9 of the invention;
10

11 FIG. 4C is an illustration of an serial multiplier, according to an embodiment of the
12 invention; and
13

14 FIG. 4D is an illustration of a micro-sequenced multiplier, according to an
15 embodiment of the invention.
16

17 DETAILED DESCRIPTION OF A PREFERRED EMBODIMENT 18

19 In the following detailed description, numerous specific details are set forth in
20 order to provide a thorough understanding of the present invention. However, it will be
21 apparent to one of ordinary skill in the art that these specific details need not be used to
22 practice the present invention. In other instances, well known structures, interfaces, and
23 processes have not been shown in detail in order not to unnecessarily obscure the present
24 invention.
25

26 An implementation of an algorithm is, generally, conceived to be a self-consistent
27 sequence of steps leading to a desired result. These steps are those requiring physical
28 manipulations of physical quantities. Usually, though not necessarily, these quantities take
29 the form of electrical or magnetic signals capable of being stored, transferred, combined,
30 compared, and otherwise manipulated. It proves convenient at times, principally for

1 reasons of common usage, to refer to these signals as bits, values, elements, symbols,
2 characters, terms, numbers or the like. It should be kept in mind, however, that all of these
3 and similar terms are to be associated with the appropriate physical quantities and are
4 merely convenient labels applied to these quantities.
5

6 In another embodiment, the present invention also relates to the apparatus for
7 performing these operations. This apparatus may be specially constructed for the required
8 purposes or it may comprise a general purpose computer as selectively activated or
9 reconfigured by a computer program stored in the computer. The algorithms presented
10 herein are not inherently related to a particular computer system or other apparatus. In
11 particular, various general purpose computer systems may be used with computer
12 programs written in accordance with the teachings of the present invention, or it may
13 prove more convenient to construct more specialized apparatus, to perform the required
14 method steps. The required structure for such machines will be apparent from the
15 description given below.
16

17 In sum, the present invention preferably is implemented for practice by a
18 computer. For example, a source code expression of the present invention is input to the
19 computer to control operations therein. It is contemplated that a number of source code
20 expressions, in one of many computer languages, could be utilized to implement the
21 present invention. A variety of computer systems can be used to practice the present
22 invention, including, for example, a personal computer, an engineering workstation, an
23 enterprise server, etc. The present invention, however, is not limited to practice on any one
24 particular computer system, and the selection of a particular computer system can be made
25 for many reasons.
26

27 In one embodiment, the invention relates to a compiler which performs mid-level
28 (target dependent) optimizations in order to make speed/area tradeoffs on an implemented
29 circuit. The choices are target-dependent, but can be directed by a set of parameters
30 determined by the target architecture.

FIG. 1 is a block diagram illustration of a system 100, according to an embodiment of the invention. The system 100 includes an source code 102, field-programmable gate array (FPGA) or custom circuit design (such as an EDIF netlist) 104, a hardware complier 106, a processor 108 and a memory device 110.

The source code 102 typically includes of programming statements that are created by a programmer in order to take the design illustrated in the programming statements and map it into a predetermined architecture. The elements of the design are set forth in the mechanics of the programming language. The programming statements (not shown) are generally created with a text editor or visual programming tool and saved in a hardware description file (not shown). Typically, the source code 102 is a programmed in a high level language, such as C or Java. In the preferred embodiment of the invention, the source code 102 can be any known programming language. The source code 102 typically represents a plurality of computations or functions units. These computations can be routines, sub-routines or other computational statements that can be represented by the programming language.

The source code 102 is input into the hardware complier 106. The hardware complier 106 runs on the processor 108 and compiles the source code 102 into configuration instruction sets which indicates the layout of the predetermined architecture. The configuration instruction set may then be stored in the memory device 110. Also, the user may input criteria into the hardware complier 106. The criteria may be the desired speed and area constraints, as well as the relative importance of each. Also, the desired circuit power and maximum time needed to complete a computation are other criteria that may be entered. The hardware compiler 106 will be discussed with greater detail with regards to FIGS. 2-3.

If the target of the compilation is an FPGA, then generated configuration bits are communicated to the field-processor gate array (FPGA) 104. The FPGA 104 is an

1 integrated circuit device that can be programmed in the field after manufacture. A
2 configuration instruction set (which is a netlist for a custom circuit or configuration
3 instructions for an FPGA) is output from the hardware compiler 106 for reconfiguring
4 FPGA 104 in order to generate the desired circuit. These devices are well known in the
5 art. If the target of the compilation is a custom circuit, the generated netlist representing
6 the circuit can then be further processed to create a custom integrated circuit.

7
8 FIG. 2 is a flow-chart illustrating the method 200 carried out by the hardware
9 compiler 106. The method 200 includes parsing the source code 210, performing an
10 optimization process 220, synthesis 230, and generating hardware realization 240. The
11 source code 102 is input into the hardware compiler 106 and a set of configuration
12 instructions e.g., configuration bits (for an FPGA target), or netlist (for a custom circuit
13 target) are output from the compiler 106. However, one of ordinary skill in the art can
14 appreciate that the hardware compiler 106 can be designed to accept various types of
15 sequential source program instructions, such as, interactive online commands, markup
16 tags, or some other defined interface.

17
18 In step 210, the input source code 102 is parsed and translated into an internal
19 representation according to a known method. The source code 102 typically describes an
20 application or algorithm to be implemented in hardware. The parsing step 210 may analyze
21 the source code in order to generate a directed hyper-graph internal representation (IR) that is
22 pure data flow. Typically, this process is performed by reversing the source code into an IR
23 that represents computations. This is done by using a combination of known techniques in
24 the art, such as static single assignment form (SSA), predication, loop unrolling and array
25 representation. The compiler 106 also performs a dependence analysis of the computation to
26 determine the time and order of execution. This is performed in a manner that is known in the
27 art. This may be accomplished by using a combination of static single assignment form,
28 predication, loop unrolling and other well-known compiler translation techniques.

1 The IR is optimized in steps 220, 225 and 235. The optimization process encompasses
2 high-level, mid-level and low-level optimization schemes. The optimization processes
3 generates a plurality of configuration instruction sets.
4

5 The high-level optimization 220 is performed by repeatedly traversing the internal
6 representation and using patterns to detect and apply transformations that reduce the size
7 of the IR. The transformations used by the high-level optimization process are, for
8 example, dead code elimination, constant folding, common sub-expression elimination,
9 logical/arithmetic identity reductions, constant propagation and strength reductions. These
10 transformations reduce the code to predicate expressions that represent the function units.
11 These optimizations are target-independent, which indicates that the knowledge of the
12 target (FPGA or custom circuit) is not needed to implement the optimizations.
13

14 Mid-level optimizations 225 allow the compiler to performs speed and time
15 tradeoffs for implementing the application operations in hardware. The mid-level
16 optimization process generates a plurality of configuration instruction sets. A
17 configuration instruction set is based on a user defined criteria. For example, circuit size,
18 desired speed, and circuit power can be defined as a user defined criteria. The
19 implementation of the hardware can be the hardware that embodies the fastest speed,
20 smallest size, or uses least amount of size. One of the plurality of configuration
21 instruction sets is selected based on the user defined criteria. The mid-level optimization
22 process will be explained in greater detail with regards to FIGS. 3 and 4.
23

24 Before the low-level optimizations can be performed, synthesis step 230 is
25 predetermined to translate the architectural and functional descriptions of the design,
26 represented typically by a dataflow graph, to a lower level of representation of the design,
27 such as logic-level and gate level descriptions. Typically, the lowest level of
28 representation is the selected configuration instruction set that is then used by the
29 placement and routing software to physically configure the hardware with the specified
30 components, as shown in step 240.

1
2 In step 235, Low-level optimizations are performed on the selected configuration set.
3 Low-level optimizations are, typically, optimizations which must be custom crafted for each
4 target platform. These optimizations are target-dependent. These optimizations can be crafted
5 for the individual target. These optimizations can be, for example, lookup table
6 (LUT)/register merging, register/LUT merging and LUT combining.
7

8 In step 240, the hardware realization is generated using the selected configuration
9 instruction set. The hardware realization may be an FPGA or customized hardware device.
10

11 FIG. 3 is a flowchart illustrating the process 300 of making mid-level
12 optimizations in the hardware compiler 106, according to an embodiment of the
13 invention. Mid-level optimizations allow the compiler 106 to make speed/area tradeoffs
14 in the implemented circuit. A plurality of optimizations are automatically performed until
15 the user-defined criteria are satisfied. The user has specified these criteria before the
16 process begins. One of ordinary skill can appreciate that FIG. 3 is a representation of one
17 such iteration.
18

19 The choices made in the mid-level optimization process 300 are target-dependent,
20 such that the optimizations can be directed by a set of parameters determined by the target
21 architecture. The mid-level optimization process 300 comprises the steps of analyzing the
22 internal representation (IR) to determine computations on the critical path as shown in
23 step 310, determine computations with a lot of slack as shown in step 320, determine
24 computations amenable to resource sharing as shown in step 330, generating the most
25 efficient design implementation as shown in step 340, and determining whether the pre-
26 defined criteria has been met in step 345.
27

28 In step 310, the mid-level optimization process 300, determines which computations
29 are on the critical path. The critical path can be a plurality of computations being carried out
30 in parallel. Typically, the process 300 will analyze the IR to look for software loops and other

1 types of repetitive code structures. For example, a loop may have one computation of $C=A*B$
2 followed by and another computation $A=C+D$. Both the multiplication and addition
3 operations are on the critical path, because the each iteration of the repetitive computation
4 cannot complete until both operations of the previous are complete. However, there may be
5 other computations in parallel.

6
7 The dependence between operations are then analyzed to determine the amount of
8 "slack" of each operation in the IR, as shown in step 320. Slack is defined as the difference
9 between the time available to complete an operation (without slowing down the speed of the
10 overall circuit) and the time that a particular implementation of that operation requires to
11 execute. For example, if the implementation is a multiplication circuit, then the slack is
12 defined as the difference between the minimum time for the system to complete a
13 multiplication operation and a particular implementation of the a multiplier circuit.

14
15 If the IR has a large amount of slack, then a slower, smaller implementation of that
16 operation could used, reducing the area of the generated circuit, without slowing down the
17 overall speed of the generated circuit. If the IR has a negative or zero amount of slack, then a
18 faster (and probably larger) implementation might provide faster overall execution. The
19 identified slack values for operations are later used to make optimization decisions in the
20 implementation of the element.

21
22 In step 330, the optimization process 300 identifies those computations which are
23 amenable to resource sharing. The optimization process 300 identifies operations that require
24 more computational resources than are available in the target platform. These computations
25 can be transformed so that the expensive resources are time-shared by several independent
26 operations. However, this process tends to slow down the speed of the computation but it is
27 offset by the corresponding reduction in area.

28
29 In step 340, a plurality of configuration instruction sets are generated. The user has
30 at least one predefined criteria, which are used to select the a configuration instruction set.

1
2 The mid-level optimizations allow the compiler to make tradeoffs in the
3 implemented circuit. Micro-pipelining, digit-serial arithmetic and micro-sequencing can
4 be employed in any one of the multiple optimized hardware realizations. In programs
5 containing no feedback, it is possible to insert additional register stages in the synthesized
6 circuit that can increase the clock rate as well as the latency and therefore micro-
7 pipelining can increase the throughput. If an operation is not pipelineable, it is possible
8 that a digit-serial implementation of the operation could lead to area-efficient
9 implementation. The latency would be large, but the overall throughput could be quite
10 high. Computations that require more computational resources than are available in the
11 target platform can sometimes be transformed so that the expensive resources are time-
12 shared by several independent computations. Micro-sequencing may slow down the
13 overall computations, but benefits from the reduction in area.

14
15 In step 340, the output of steps 310-330, may include a plurality of configuration
16 instruction sets, wherein each instruction set is operable to generate a different optimized
17 hardware realization. These steps 310-330 may use the user-defined criteria to generate
18 different configuration instruction sets.

19
20 In step 345, the configuration instructions are simulated. For example, a configuration
21 instruction set is transmitted to a simulator. The simulator generates a simulation of the
22 hardware realization based on the configuration instruction set. This maybe performed for all
23 of the generated configuration instructions sets.

24
25 The simulator is used to test the hardware, based on the predetermined criteria, as
26 shown in step 350. The simulator may test the hardware realization for speed, circuit power,
27 and time to execute a predetermined operation. The simulator may also examine the size of
28 the hardware realization. Typically, this operation is performed outside the compiler.
29

1 In step 360, a configuration instruction set is selected. The compiler receives the
2 results from the simulation and automatically selects the configuration instruction set that
3 closely matches the user's criteria.

4
5 FIGS. 4A, 4B, 4C and 4D illustrate block diagrams of multiple circuits, in accordance
6 with the invention. The FIGS. (4A-4D) illustrate three different ways that the following code,

7 int a, b, c;

8 while (1)

9 c=a*b;

10 can be implemented using the mid-level optimization process 300. One of ordinary skill can
11 readily appreciate that the above code and followings illustrations are for illustrative purposes
12 only and not meant to limit the scope of the invention in any way. The above code illustrates a
13 multiplication operation. FIGS. 4(A-D) illustrates an example of the different types of
14 multipliers that can be implemented based on the user's predefined criteria.

15
16 As stated above, with regards to FIG.3, the mid-level optimization process 300
17 determines speed and area tradeoff in the implemented circuit. The process 300 generates
18 a plurality of configuration instruction sets representing a plurality of hardware
19 realizations (design spaces). These plurality of configuration instructions sets are
20 simulated and tested and compared with the user defined criteria. A configuration
21 instruction set is selected based on the output of this comparison. As stated above, the
22 goal could be, for example, size of the circuit, speed, total area of the circuit, and circuit
23 power. The implementation which exemplifies the user's criteria is then selected and the
24 associated configuration instruction set is transmitted to the FPGA or custom circuit 104.

25
26 FIGS. 4(A-D) illustrates a plurality of hardware realizations that can be implemented
27 from the above software. Each of the plurality of realizations has specific area, speed and
28 power characteristics.

1 FIG. 4A illustrates an unpipelined multiplier circuit 400 which can be implemented by
2 optimization process. The unpipelined multiplier circuit 400 operates by receiving inputs via
3 operands, a and b every clock cycle ($\Pi=1$) and produces one result (c) at the end of the clock
4 cycle. The unpipelined multiplier 400 generally consumes a great deal of area and is slow
5 because of the large number of logic levels required in its implementation.

6
7 FIG. 4B illustrates a pipelined multiplier 410 which can be implemented by from the
8 above code. The pipelined multiplier 410 is slightly larger than the unpipelined multiplier
9 400, but has a higher clock speed because of the staging registers, as well as increased
10 latency. FIG. 4C illustrates a digit-serial version (such as a semi-systolic, bit serial multiplier)
11 of the multiplier 420 represented by the above code. The digit-serial multiplier 420 requires a
12 small amount of area, but can input new operands a and b every N cycles, which increases the
13 latency.

14
15 FIG. 4D illustrates a micro-sequenced multiplier 440 implementing a time-sharing
16 capability between function units. The multiplier 440 illustrates how an expensive function
17 unit can be time shared between two independent computations by multiplexing the inputs.

18
19 The above multiplier implementations reflect an example of the tradeoffs in the
20 optimization implementation. One of ordinary skill can appreciate that the best
21 implementation or the implementation selected by the user will depend on a plurality of
22 factors such as the nature of the program, the implementation or problem to be solved.

23
24 What has been described and illustrated herein is a preferred embodiment of the
25 invention along with some of its variations. The terms, descriptions and figures used herein
26 are set forth by way of illustration only and are not meant as limitations. Those skilled in the
27 art will recognize that many variations are possible within the spirit and scope of the
28 invention, which is intended to be defined by the following claims -- and their equivalents --
29 in which all terms are meant in their broadest reasonable sense unless otherwise indicated.